

MySql精讲-曾庆锋

www.toceansoft.com

Sql 知识纲要

1. 了解sql
2. 使用mysql
3. 检索数据
4. 排序检索数据
5. 数据过滤
6. 用通配符进行过滤
7. 创建计算字段
8. 数据处理函数
9. 汇总函数
10. 分组数据
11. 子查询
12. 表联结
13. 合并查询
14. 插入数据
15. 更新删除数据
16. 创建表和操作表

第1章 了解Sql

1.1 准备

- 下载sql脚本（分发给学员）
- Create.sql 为建表语句
- Populate.sql为插入数据语句

1.2 脚本说明

- Vendors表 (供应商)

列	说明
Vend_id	唯一供应商ID
Vend_name	供应商名
Vend_address	供应商地址
Vend_city	供应商地址
Vend_state	供应商的州
Vend_zip	供应商邮编
Vend_country	供应商的国家

1.2 脚本说明

- Products表 （产品）

列	说明
Prod_id	惟一产品ID
Vend_id	产品供应商ID（关联到vendors表中的vend_id）
Prod_name	产品名
Prod_price	产品价格
Prod_desc	产品描述

1.2 脚本说明

- Customers表 (顾客)

列	说明
Cust_id	顾客惟一ID
Cust_name	顾客名
Cust_address	顾客的地址
Cust_city	顾客的城市
Cust_state	顾客的州
Cust_zip	顾客的邮编
Cust_country	顾客的国家
Cust_contact	顾客的联系名
Cust_email	顾客的Email

2.脚本说明

- Orders表 （订单）

列	说明
Order_num	惟一订单号
Order_date	订单日期
cust_id	订单顾客ID(关系到customers表的cust_id)

1.2 脚本说明

- Orderitems表（订单明细表-用于存储每个订单中的实际物品）

列	说明
Order_num	惟一订单号（关联到orders表的order_num）
Order_item	订单物品号（在某个订单中的顺序）
Prod_id	产品ID(关联到products表的prod_id)
quantity	物品数量
Item_price	物品价格

1.2 脚本说明

- Productnotes表（特定产品的注释）

列	说明
Note_id	惟一订单号（关联到orders表的order_num）
Prod_id	产品ID(关联到products表的prod_id)
Note_date	增加注释的日期
Note_text	注释文本

1.3 命令行导入sql脚本

1 打开命令终端

程序->运行->cmd

2 连接数据库

mysql -u 账号 -p 密码

例:

mysql -u root -p 123456

3 创建数据库

CREATE DATABASE 数据库名字 DEFAULT CHARECTSET 编码

例:

CREATE DATABASE crashcourse DEFAULT CHARSET utf8

4 使用刚创建的数据库

USE crashcourse

5 导入脚本

source create.sql

source populate.sql

6 检查是否导入

mysql> show tables;

1.4 命令行导出sql脚本

1 打开命令终端

程序->运行->cmd

2 导出建表语句和数据

mysqldump -u 账号 **-p** 数据库 **>** 脚本文件

例：

mysqldump -u root -p test > mytest.sql

1.5 工具导入、导出sql脚本

- 请看实操

1.6 数据库基础

- 什么是数据库
- 表
- 列和数据类型
- 行
- 主键
- 什么是SQL

第2章使用mysql

2.1连接数据库

本地数据库

> ***mysql -u root -p***

输入密码

远程数据库

> ***mysql -h 主机名(IP) -u 账号 -p***

例:

> ***mysql -h 192.168.1.115 -u root -p***

输入密码

2.2选择数据库

USE 数据库名

例:

USE crashcourse;

2.3了解数据库和表

1）查看所有数据库

>SHOW DATABASES;

2) 查询某个数据库下的所有表

>SHOW TABLES;

3）查看某个表下的所有列

>SHOW COLUMNS FROM 表名

例:

>SHOW COLUMNS FROM customers;

另一种简写形式DESC 表名

例:

DESC customers;

4）查询服务器状态信息

SHOW STATUS;

5) 显示创建数据库的sql语句

SHOW CREATE DATABASE 数据库名

例:

SHOW CREATE DATABASE shop;

2.3了解数据库和表

6) 显示创建表的语句

SHOW CREATE TABLE 表名

例：

SHOW CREATE TABLE customers

7) 显示授予用户的安全权限

SHOW GRANTS ;

8) 显示服务器错误

SHOW ERRORS

9) 显示警告消息

SHOW WARNINGS

10) 显示允许的**SHOW**语句

HELP SHOW;

第3章检索数据

3.1检索单个例

```
> SELECT prod_name FROM products;
```

说明：

1）未排序数据

以上检索的数据是未给过排序的。 可能是按插入的顺序，也可 能不是。只要返回相同数目的行 就是正常的

2）结束sql语句

以分号(;)结束

3）sql语句大小写

不区分大写写

但最好按一定规则，

比如关键字大小，表名列名小写

4）写多行更容易阅读和调试

3.1检索多个例

```
SELECT prod_id, prod_name,
   prod_price
FROM products;
```

3.3检索所有例

SELECT * FROM products;

说明：

号是通配符，代表所有例。除非 确实需要表中的每个列，否则最好不要用 ,虽然省事，但检索不需要的列通常会降低 检索和应用程序的性能

3.4去掉相同的行

SELECT vend_id FROM products;

有重复数据

SELECT DISTINCT vend_id FROM products;

DISTINCT 关键字应用于所有列而不是第一个例

3.5限制结果

```
SELECT prod_name FROM products
  LIMIT 5;
```

带一个值，总是从第一行开始，查询5行

```
SELECT prod_name FROM products
  LIMIT 5,5;
```

带2个值，第一个值代表从第5行开始，第2个值命名要检索多少行

注：LIMIT 可用来做分页

3.6 使用完全限定表名

使用完全限定表名

```
SELECT products.prod_name FROM
  crashcourse.products;
```

第4章 排序检索数据

4.1排序数据（ORDER BY ）

```
SELECT prod_name
FROM products;
```

说明：可能是数据最初添加到表中的顺序，关系数据库设计理论认为，

如果不明确规定排序顺序，则不 应该假定检索出的数据的顺序有 意义

VS 对比

```
SELECT prod_name
FROM products
```

4.2按多个例进行排序

```
SELECT prod_id, prod_price,
   prod_name
FROM products
ORDER BY prod_price, prod_name;
```

说明：先按prod_price排序，如果prod_price相同，则再按prod_name的字典顺序排序

4.3 指定排序方向

可升序(ASC),可降序（DESC）

例：按价格以降序排序(最贵的排在最前面)

```
SELECT prod_id, prod_price,
prod_name
FROM products
ORDER BY prod_price DESC
```

例：对产品价格进行升序排序，再对产品名进行降序排序

```
SELECT prod_id, prod_price,
```

4.4 ORDER BY 和 LIMIT组合

例：查找最昂贵物品的值

```
SELECT prod_price
FROM products
ORDER BY prod_price DESC
LIMIT 1;
```

说明：

ORDER BY 子句的位置

保证在**FROM**子名之后，**LIMIT**子名之前

第5章过滤数据

5.1使用WHERE子句

例：查询价格等于2.50的商品

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price = 2.50;
```

说明：

1）WHERE子句的位置

如果使用ORDER BY 和 WHERE 子句，应该让ORDER BY 位于WHERE 之后

5.2 WHERE 子句操作符

5.2.1

操作符	说明
=	等于
<>	不等于
!=	不等于
<	小于
<=	小于等于
>	大于
>=	大于等于
BETWEEN	在指定的2个值之间

5.2 WHERE 子句操作符

5.2.2 检查单个值

例1：列出产品名为fuses的产品

```
SELECT prod_name, prod_price
FROM products
WHERE prod_name='fuses'
```

例2：列出价格小于10美元的所有产品

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price < 10;
```

例3：例出价格大于等于10美元的产品

5.2 WHERE 子句操作符

❖ 5.2.3 不匹配检查

例：列出不是由供应商1003制造的所有产品

```
SELECT vend_id,prod_name
FROM products
WHERE vend_id <>1003
```

```
SELECT vend_id,prod_name
FROM products
WHERE vend_id != 1003
```

5.2 WHERE 子句操作符

❖ 5.2.4 范围检查

例：列出价格在5美元和10美元之间的产品

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price BETWEEN 5 AND 10;
```

```
SELECT prod_name, prod_price
FROM products
WHERE prod_price>= 5 AND prod_price<=10;
```

5.3 空值检查

- IS NULL 和 IS NOT NULL

例1:列出没有价格(不是为0)的所有商品

```
SELECT prod_name
FROM products
WHERE prod_price IS NULL
```

例2：列出email不为空的顾客

```
SELECT cust_id
FROM customers
WHERE cust_email IS NOT NULL
```

5.4组合WHERE子句

5.4.1 AND操作符

例：检索由供应商1003制造且价格小于等于10美元的所有产品和价格

```
SELECT prod_id, prod_price, prod_name
FROM products
WHERE vend_id = 1003 AND prod_price <= 10;
```

5.4组合WHERE子句

5.4.2 AND操作符

例：检索由供应商1003制造且价格小于等于10美元的所有产品和价格

```
SELECT prod_id, prod_price, prod_name
FROM products
WHERE vend_id = 1003 AND prod_price <= 10;
```

5.4组合WHERE子句

❖ 6.4.3 OR操作符

例：检索由供应商1002或者1003制造的所有产品

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id = 1002 OR vend_id = 1003;
```

5.4组合WHERE子句

5.4.4 计算次序

例： 列出价格大于等于10美元且由1002或1003制造的所有的产品

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id =1002 OR vend_id= 1003 AND prod_price >= 10;
```

说明：

返回的行中有我们不想要的结果，为什么呢？原因是AND的优先级高于OR,MYSQL理解成查询供应商为1002 或者供应商为1003且价格大于等于10 的产品

```
SELECT prod_name, prod_price
FROM products
WHERE (vend_id =1002 OR vend_id= 1003) AND prod_price >= 10;
```

说明 ：

这回得到了我们想要的结果。 任何时候使用具有AND和OR 操作符的WHERE子句，都应该 使用圆括号明确的分组操作符， 不要过分依赖默认的计算次序 即使它确实是你想要的东西也是 如此。圆括号没什么坏处，它能消除歧义

5.5 IN操作符

IN操作符用来指定条件范围，范围中的每个条件都进行匹配

例：检索供应商1002和1003供应的商品

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id IN (1002, 1003)
ORDER BY prod_name;
```

对比：

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id =1002 OR vend_id = 2003
ORDER BY prod_name;
```

说明：

2条Sql语句完成相同的功能，为什么要使用IN操作符？在使用长的合法选项清单中时，IN操作符的语法更清楚更直观在使用IN 时，计算的次序更容易管理（使用的操作符更少）IN 操作符一般比OR操作执行更快IN 的最大优点是可以包含其他SELECT 语句，使得能更动态的建立WHERE子句

5.6 NOT操作符

NOT IN

NOT BETWEEN

NOT EXISTS

第6章用通配符进行过滤

6.1 LIKE操作符

6.1.1 百分号(%)通配符

例1：找出所有以词jet起头的产品

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE ‘jet%’
```

说明：

%告诉MySql接受jet之后的任意字符，不管它有多少字符

例2：找出产品名含有anvil字符的产品

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '%anvil%';
```

说明：

注意尾空格，假设anvil后面有一个或多个空格 时，则 子名WHERE prod_name LIKE ‘%anvil ’将不会匹配它们。 解决的办法是最后附加一个%,还有就是使用去空格函数。 注意NULL，虽然%可以匹配任何东西，但不能匹配NULL.

6.1 LIKE操作符

6.1.2 下划线(_)通配

只匹配一个字符,不能多也不能少

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE '_tom anvil'
```

6.2 使用通配符技巧

1）不要过度使用通配符，如果其他操作能达到相同的目的，应该使用其他操作符

2）在确实需要使用通配符时，除非绝对必要，否则不要把它们用在搜索模式的开始处。这样搜索起来很慢

3）注意通配符的位置。如果放错地方，可能不会返回想要的数据

第7章 创建计算字段

7.1 计算字段

现实需求

1）如果想在一个字段中既显示公司名， 又显示公司的地址，但这2个信息一般 包含在不同的例中

2）城市、州和邮编存储在不同的列中， 但邮件标签打印却需要把它们作为一个恰当的字段检索出来

3）求物品的总价格

4）计算选中物品的平均价格

7.2 拼接字段

- 使用Concat函数

例：要生成一个供应商报表，需要在供应商的名字按照‘供应商名（供应商地址）’这样的格式列出来，这个时候需要将vend_name和vend_country拼起来

```
SELECT
   Concat(vend_name,' (',vend_country,')')
FROM vendors
ORDER BY vend_name;
```

说明：其它数据库如Oracle采用||拼接

7.3 执行算术计算

- 例：检索订单号为20005中的所有物品，并且计算每个产品的总体(产品数量*单体)
- SELECT prod_id, quantity, item_price, quantity*item_price AS expnded_price
- FROM orderitems
- WHERE order_num = 20005;

7.4 测试运算

省略FROM子句

```
SELECT Trim('abc')
SELECT Now()
SELECT 3*2
```

第8章使用数据处理函数

8.1 函数

函数：用来处理数据

函数在数据库之间可移值性差，所以添加注释

8.2 常用的文本处理函数

函数	说明
Left()	返回串左边的字符
Right()	返加串右边的字符
Length()	返回串的长度
Locate()	找出串的一个子串
Lower()	将串转换成小写
lTrim()	去掉串左边的空格
RTrim()	去掉串右边的空格
Substring()	返回子串的字符
Upper()	将串转换成大写

8.3日期和时间处理函数

函数	说明
AddDate()	增加一个日期（天、周等。）
AddTime()	增加一个时间（时，分）
CurDate()	返回当前日期
CurTime()	返回当前时间
Date()	返回日期时间的日期部分
DateDiff()	计算2个日期之差
DateAdd()	高度灵活的日期运算函数
Date_Format()	返回一个格式化的日期或时间串
Day()	返回一个日期的天数部分
DayOfWeek()	对于一个日期，返回对应的是星期几
Hour()	返回一个时间的小时部分

8.3日期和时间处理函数

函数	说明
Minute()	返回一个时间的分钟部分
Month()	返回一个日期的月份部分
Now()	返回当前日期的时间
Second()	返回一个时间的秒部分
Time()	返回一个日期的时间部分
Year()	返回一个日期的年份部分

8.4数值处理函数

函数	说明
Abs()	返回一个数的绝对值
Cos()	返回一个角度的余弦
Exp()	返回一个数的指数值
Mod()	返回除操作的余数
Pi()	返回圆周率
Rand()	返回一个随机数
Sin()	返回一个角度的正弦
Sqrt()	返回一个数的平方根

第9章汇总数据（聚集函数）

9.1 AVG()-返回某列的平均值

例1：查询所有产品的平均价格

```
SELECT AVG(prod_price) avg_price
FROM products;
```

例:查询供应商1003提供的所有商品的价格

```
SELECT AVG(prod_price) avg_price
FROM products
WHERE vend_id = 1003;
```

说明：

AVG()只用于单个列，并且会忽略列值为NULL

9.2 COUNT()-返回某列的行数

有2种使用方式

1) 使用COUNT(*)对表中行的数目进行计数，不管表列中包含的是空值(NULL) 还是非空值

2) 使用COUNT(column) 对特定列中具有值的行数进行计数，忽略NULL值

例：查询costomers表中的客户总数

```
SELECT COUNT(*)  num_cust
FROM customers;
```

例：只对具有电子邮件地址的客户计数

9.3 MAX()-返回某列的最大值

例：列出最贵的物品的价格

```
SELECT MAX(prod_price) max_price
FROM products;
```

说明：

对非数值数据使用MAX()，虽然MAX()一般用来找出最大的数值或日期值，但MYSQL允许将它用来返回任意列中 的最大值，包括返回文本列中的最大值。在用于文本数据时，如果数据按相应的列排序 ，则MAX()返回最一行。 MAX()函数忽略列值为NULL的行

8.4 MIN()-返回某列的最小值

例：查询最便宜商品的价格

```
SELECT MIN(prod_price) min_price
FROM products;
```

8.5 SUM()-返回某列值之和

例：检索所订单20005订购物品的总数

```
SELECT SUM(quantity) items_ordered
FROM orderitems
WHERE order_num = 20005;
```

例：求订单20005的总的金额

```
SELECT SUM(item_price*quantity) total_price
FROM orderitems
WHERE order_num = 20005;
```

第10章分组数据

10.1数据分组

例：查询供应商1003提供的产品数目

```
SELECT count(*) num_prods
FROM products
WHERE vend_id = 1003;
```

- 例：查询每个供应商及它们提供产品的数目
- 例：查询只提供了单项产品的供应商所提供的产品
- 例：列出提供10个以上产品的供应商

10.2创建分组

上面后面三个例子可以通过分组解决

例：查询每个供应商及它们提供产品的数目

```
SELECT vend_id, COUNT(*) num_prods
FROM products
GROUP BY vend_id;
```

说明：

GROUP BY 对vend_id排序并分组数据

10.3 GROUP BY的重要规定

1）GROUP BY 子句可以包含任意数目的列，这使 得能对分组进行嵌套，为数据分组提供更细致的控制

2）如果在GROUP BY 子句中嵌套了分组，数据将在最后规定的分组上进行汇总，换句话说，在建立分组时，指定的所有列都 一起计算（所以不能从个别的列取回数据）

3）GROUP BY 子句中列出的每个列 都必须是检索列或有效的表达式（但）不能是聚集函数。如果在SELECT 中使用 表达式，则必须 在GROUP BY 子句中指定相同 的表达式，不能使用别名。

4）除聚集计算语句外，SELECT 语句中的每个列都必须在GROUP BY 子句中给出

5）如果分组列中具有NULL值，则NULL值将作为一个 分组返回。如果列中有多行NULL值，它们将分成一组

6）GROUP BY 子句必须出现在WHERE子句之后，ORDER BY 子句之前

10.4过滤分组

例：列出至少有两个订单的顾客

```
SELECT cust_id, COUNT(*) orders
FROM orders
GROUP BY cust_id
HAVING COUNT(*) >= 2;
```

HAVING和WHERE的差别：

- **WHERE过滤的是指定的行而不是分组数据**
- **HAVING子句用来过滤分组**
- **WHERE 在数据分组前进行过滤，HAVING在数据分组后进行**

10.4过滤分组

HAVING 和 WHERE同时使用

例：列出具有2个以上，价格为10以上的产品的供应商

```
SELECT vend_id, COUNT(*) num_prods
FROM products
WHERE prod_price >= 10
GROUP BY vend_id
HAVING COUNT(*) >=2;
```

10.5分组和排序

例：检索总计订单价格大于等于50的订单的订单号和总计订单价格，并按总计订单价格排序

```
SELECT order_num, SUM(quantity*item_price) ordertoal
FROM orderitems
GROUP BY order_num
HAVING SUM(quantity*item_price) >=50
ORDER BY ordertotal;
```

10.6 SELECT 子句顺序

子句	说明	是否必须使用
SELECT	要返回的列或表达式	是
FROM	从中检索数据的表	仅在从表选择数据时使用
WHERE	行级过滤	否
GROUP BY	分组说明	仅在按组计算聚集时使用
HAVING	分组过滤	否
ORDER BY	排序	否
LIMIT	要检索的行数	否

第11章使用子查询

11.1子查询

- 子查询：嵌套在其它子句里的查询

10.2 利用子查询进行过滤

需求：列出订购物品TNT2的所有客户，应该怎么样检索

没有子查询

1）检索包含物品TNT2的所有订单的编号

```
SELECT order_num
FROM orderitems
WHERE prod_id = 'TNT2';
```

2）检索具有前一步骤列出的订单编号的所有客户ID

```
SELECT cust_id FROM orders
WHERE order_num IN (20005,20007);
```

10.2 利用子查询进行过滤

用子查询

```
SELECT cust_id, cust_name
FROM customers
WHERE cust_id IN (SELECT cust_id
                  FROM orders
                  WHERE
    order_num IN ( SELECT order_num

                   FROM orderitems

                   WHERE prod_id = 'TNT2'))
```

10.3作为计算字段使用子查询

需求：显示customers表中每个客户的订单总数

```
SELECT cust_name,
       cust_state,
       (SELECT COUNT(*)
        FROM orders
       WHERE orders.cust_id =
  customers.cust_id) AS orders
FROM customers
ORDER BY cust_name;
```

- 相关子查询：涉及外部查询的子查询

好的步骤

逐渐增加子查询来建立查询

首先建立和测试最内层的查询

然后用硬编码数据建立和测试外层查询，

并且仅在确认它正常之后才嵌入子查询

第12章联结表

12.1 联结

❖12.1.1 关系表

现实世界中的例子

假如有一个包含产口目录的数据库表，其中每种类别的物品占一行。

对于每种物品要存储的信息包括产品描述和价格，以及生产该 产品

的供应商信息。（看图）

现在，假如有由同一供应商生产的多种物品，

12.1 联结

12.1.1 关系表

如果存储在同一个表中，会出现以下问题

1）因为同一供应商生产的每个产品的供应商信息都是相同的， 对每个产品重复此信息既浪费时间又浪费空间

2）如果供应商信息改变（例如供应商搬家或电话号码变动），只需改动一次即可

3）如果有重复数据（即不同产品对应同一个供应商），很难保证每次输入该数据的方式相同。不一致的数据在报表中很难利用

基于以上三点，我们有理由将他们分类存储

12.1 联结

12.1.1 关系表

相同数据出现多次决不是一件好事，这是关系数据库设计的基础。

关系表的设计就是要保证把信息按一定规则分解成多个表，各表之间能过某些常用的值（即关系设计中的关系）互相关联

在刚才的例子中，可建立2个表

vendors表-供应商

products表-商品

2个表之间通过外键关联

12.1 联结

12.1.2 为什么要使用联结

联结使得我们能够一次查询多个表，返回多个表的数据

12.2 创建联结

例：查询所有的商品及各产品的供应商信息

```
SELECT vend_name, prod_name, prod_price
FROM vendors , products
WHERE vendors.vend_id=products_vend_id;
```

12.3 WHERE语句的重要性

在联结表时，实际上时交第一个表中的每一行与第二个表的每一行配对WHERE语句作为过滤条件，返回的结果只包含那些匹配给定条件的行没有WHERE子句，第一个表中的每个行将与第二个表中的每个行配对

请看下面例子：

```
SELECT vend_name,prod_name, prod_price
FROM vendors, products
ORDER BY vend_name, prod_name;
```

12.4内部联结（INNER JOIN）

等值联结：基于2个表之间的相等测试，也称为内部联结

```
SELECT vend_name, prod_name, prod_price
FROM vendors INNER JOIN products
ON vendors.vend_id = products.vend_id;
```

- 这是ANSI SQL规范首选INNER JOIN语法

12.5联结多个表

例：显示编号为20005的订单中的物品

```
SELECT prod_name, vend_name, prod_price,
  quantity
FROM orderitems, products, vendors
WHERE products.vend_id = vendors.vend_id
AND orderitems.prod_id = products.prod_id
AND order_num = 20005;
```

分析

order_num=20005过滤出orderitems表中的订单号为20005的商品

12.5联结多个表

例:显示订购产品TNT2的客户列表

- 用子查询:

```
SELECT  cust_name, cust_contact
FROM customers
WHERE cust_id IN (SELECT cust_id
                  FROM orders
                  WHERE order_num IN (SELECT order_num
                                      FROM orderitems
                                      WHERE prod_id = 'TNT2'));
```

- 利用多表连接:

```
SELECT cust_name, cust_contact
FROM customers, orders, orderitems
WHERE customers.cust_id = orders.cust_id
AND orderitems.order_num = orders.order_num
AND prod_id ='TNT2'
```

10.6使用表别名

例：显示订购产品TNT2的客户列表

- 利用多表连接：

```
SELECT cust_name, cust_contact
FROM customers, orders, orderitems
WHERE customers.cust_id = orders.cust_id
AND orderitems.order_num =
   orders.order_num
AND prod_id ='TNT2'
```

- 使用表别名后

```
SELECT cust_name, cust_contact
```

10.7自联结

例子：假如你发现某物品（其ID为DTNTR）存在问题，因此想知道生产该物品的供应商生产的其他物品是否也存在，为此查询要求先找到生产ID为DTNTR的物品的供应商，然后找出这个供应商生产的其他物品

- 第一种解决方案:子查询

```
SELECT  prod_id, prod_name
FROM products
WHERE vend_id = (SELECT vend_id FROM products
                 WHERE prod_id = 'DTNTR')
```

- 第二种方案：自连接

```
SELECT p1.prod_id, p1.prod_name
FROM products p1, products p2
WHERE p1.vend_id = p2.vend_id
AND p2.prod_id  = 'DTNTR';
```

- 基于性能考虑，一般采用自联结而不是子查询

10.8外部联结

现实需求

1）对每个客户下了多少订单进行计数，包括那些至今尚未下订单的客户

2）列出所有产品以及订购数量，包括没有人订购的产品

3）计算平均销售规模，包括那些至今尚未下单的客户

以上三个例子都要对表进行联结，并且包含了那些在相关表中没有关联行的行

外部联结：联结包含了那些在相关表中没有关联行的行，这种类型的连接称为外部联结

10.8外部联结

例：检索所有客户及其订单

- *采用内联结(等值联结)*

```
SELECT customers.cust_id, orders.order_num
FROM customers INNER JOIN orders
ON customers.cust_id = orders.cust_id;
```

发现结果中没有包含那些没下订单的客户

- *采用外联结*

```
SELECT customers.cust_id, orders.order_num
FROM customers LEFT OUTER JOIN orders
ON customers.cust_id = orders.cust_id;
```

10.8外部联结

左外联结(LEFT OUT JOIN)

从FROM子句的左边表中选择所有的行

右外联结(RIGHT OUT JOIN)

从FROM子句的右边表中选择所有的行

```
SELECT customers.cust_id, orders.order_num
FROM customers RIGHT OUTER JOIN orders
ON customers.cust_id = orders.cust_id;
```

10.9使用带聚集函数的联结

例：检索所有客户及每个客户所下的订单数

```
SELECT c.cust_name, c.cust_id,
  COUNT(o.order_num) num_ord
FROM customers c INNER JOIN orders o
ON c.cust_id = o.cust_id
GROUP BY c.cust_id;
```

对比

```
SELECT c.cust_name, c.cust_id,
  COUNT(o.order_num) num_ord
FROM customers c LEFT OUTER JOIN orders o
```

10.10使用联结和联结条件的约定

1）注意使用的联结类型。一般我们使用内部联结，但使用外部联结也是有效的

2）保证使用正确的联结条件，否则将返回不正确的数据

3）应该总是提供联结条件，否则会得出笛卡尔积

4）在一个联结中可以包含多个表，甚至对于每个联结可以采用不同的联结类型。一般先测试每个联结，再使用。

第13章 合并查询

13.1 组合查询

MySQL请允许执行多个查询，并将结果作为单个查询结果返回

需要使用组合查询的情况：

1）在单个查询中从不同的表返回类似结构的数据

2）对单个表执行多个查询，按单个查询返回数据

13.2 创建合并查询

需求：查询需要价格小于等于5的所有物品的一个列表，

而且还想包括供应商1001和1002生产的所有物品

- 不用UNION

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5
OR   vend_id IN (1001,1002);
```

- 使用UNION

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5
UNION
```

13.3 使用UNION规则

1) UNION必须由两条以上的SELECT 语句组成，语句之间用关键字UNION分隔

2）UNION中的每个查询必须包含相同的列、表达式或者聚集函数（不过每个列不需要以相同的次序列出）

3） 列数据类型必须兼容：类型不必完全相同，但必须是DBMS可以
隐含地转换的类型（例如，不同的数值类型或不同的日期类型）

13.4包含或取消重复的行

- UNION 会自动取消重复的行
- UNION ALL 返回所有匹配的行，不取重复

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5
UNION ALL
SELECT vend_id, prod_id, prod_price
FROM products
WHERE vend_id IN (1001,1002);
```

13.5对组合查询结果排序

```
SELECT vend_id, prod_id, prod_price
FROM products
WHERE prod_price <= 5
UNION ALL
SELECT vend_id, prod_id, prod_price
FROM products
WHERE vend_id IN (1001,1002)
ORDER BY vend_id, prod_price;
```

- 排序是对合并后的所有数据排序

第14章插入数据

14.1数据插入方式

1）插入完整的行
2）插入行的一部分
3）插入多行
4）插入某些查询的结果

14.2插入完整的行

不给出列

```
INSERT INTO customers
VALUES (NULL,
        'Pep E. LaPew',
        '100 Main Street',
        'Los angeles',
        'CA',
        '90046',
        'USA',
        NULL,
        NULL);
```

说明：

省略了列，**VALUES**子句中必须提供所有列的值

存储到每个表列中的数据在**VALUES**子句中给出

如果某个列没有值（如**cust_contact**）,则使用**NULL**

各个列必须以它们在表定义中出现的次序填充

cust_id是数据库自动增量，你不用给出值，但又不能省略，所以指定一个**NULL.**

省略所有列极不安全，避免使用。高度依赖表中列的定义次序。

14.2插入完整的行

指定列

```
INSERT INTO
customers(cust_name,
          cust_address,
          cust_city,
          cust_state,
          cust_zip,
          cust_country,
          cust_contact,
          cust_email
)
VALUES ( 'Pep E. LaPew',
                 '100 Main Street',
                 'Los angeles',
                 'CA',
                 '90046',
                 'USA',
                 NULL,
                 NULL);
```

说明：

列与值一一对应

即使表的结构改变，此INSERT语句仍然能正确工作。

14.3注意：

- 总是使用例的列表

  一般不要使用没有明确给出列的列表的INSERT语句。
  使用列的列表能SQL代码继续发挥作用，即使
  表结构发生了变化。

- 仔细地给出值

  不管使用哪种INSERT语法，都必须给出VALUES的正确数目。
  如果不提供列名，则必须给每个列提供一个值 。如果提供列名
  则必须给每个表列提供一个值 。如果不这样，将产生一条错误
  信息，相应的行插入不成功。

- 省略列必须满足条件

  1）该例定义为允许为NULL值（无值或空值）
  2）在表定义中给出默认值 。这表示如果不给出值，将使用默认值

- 提高整体性能

  INSERT操作可能很耗时（特别是有 很多索引需要更新时）， 而且它可能降低等待处理的SELECT语句的性能 如果果SELECT 更重要，可通过在INSERT和 INTO之间添加关键字LOW_PRIORITY降低优先级

14.4插入多个行

- 执行多个INSERT
- 执行一个INSERT插入多行

```
INSERT INTO customers(cust_name,
        cust_address,
        cust_city,
        cust_state,
        cust_zip,
        cust_country )
VALUES ( 'Pep E. LaPew',
           '100 Main Street',
           'Los angeles',
           'CA',
           '90046',
           'USA' ),
         ('M. Martian',
           '40 Galaxy way',
           'New York',
           'NY',
           '11213',
           'USA'   );
```

使用以上语法可提高INSERT的性能

10.5插入检索出的数据

从一个表中检索数据插入另一个表

```
INSERT INTO custnew(
    cust_id,
    cust_contatct,
    cust_email,
    cust_name,
    cust_address,
    cust_city,
    cust_state,
    cust_zip
    cust_country )
SELECT
    cust_id,
    cust_contatct,
    cust_email,
    cust_name,
    cust_address,
    cust_city,
    cust_state,
    cust_zip
    cust_country
FROM customers;
```

第15章更新和删除数据

15.1 更新数据(UPDATE)

- 1）更新表中所有的行

```
UPDATE customers
SET cust_email = 'elmer@fudd.com';
```

观察更新后的结果

15.1 更新数据(UPDATE)

- 2)更新表中特定的行

```
UPDATE customers
SET cust_email = 'zengqf@163.com'
WHERE cust_id =10005;
```

观察更新后的结果

- 说明：

  不要省略WHERE子句,否则会更新所有行

  在UPDATE语句中使用子查询

  IGNORE关键字

16.2 删除数据(DELETE)

- 1）从表中删除所有的行

```
DELETE FROM customers;
```

- 2）从表中删除特定行

```
DELETE FROM customers
WHERE cust_id = 10006;
```

16.3 更新和删除的指导原则

1）除非确实更新和删除所有行，否则绝对不要 使用不带WHERE子句的UPDATE和DELETE语句

2）保证每个表都有主键，尽可能在WHERE子句使用它。（可以指定各主键，多个值或值 的范围）

3）在对UPDATE或DELETE 语句使用WHERE子句前，应该先用SELECT 进行测试，保证它过滤的是正确的记录，以防编写的WHERE子句不正确

第16章创建表和操纵表

16.1 创建表

21.1.1 表创建基础

```
CREATE TABLE customers (
  cust_id      int       NOT NULL
   AUTO_INCREMENT,
  cust_name    char(50)  NOT NULL ,
  cust_address char(50)  NULL ,
  cust_city    char(50)  NULL ,
  cust_state   char(5)   NULL ,
  cust_zip     char(10)  NULL ,
  cust_country char(50)  NULL ,
  cust_contact char(50)  NULL ,
```

16.1 创建表

- 21.1.2

数据类型

char(n)

varchar(n)

int

bigint

Datetime

16.1 创建表

- 21.1.2 使用NULL值

  cust_name 在INSERT的时候必须给值

  cust_email可以不给值，或者给NULL值

16.1 创建表

❖21.1.3 主键

单列做为主键

```
PRIMARY KEY(vend_id)
```

复合主键（多列）

```
PRIMARY
KEY(order_num,order_item)
```

主键必然是惟一且不能为空

16.1 创建表

- 21.1.4 使用AUTO_INCREMENT

  即使设置了AUTO_INCREMENT也可以指一一个惟一值

  确定AUTO_INCREMENT值

  SELECT last_insert_id()

16.1 创建表

21.1.5 指定默认值

```
CREATE TABLE orderitems
(
  order_num  int          NOT NULL ,
  order_item int          NOT NULL ,
   prod_id    char(10)     NOT NULL ,
  quantity   int          NOT NULL DEFAULT 1 ,
  item_price decimal(8,2) NOT NULL ,
  PRIMARY KEY (order_num, order_item)
) ENGINE=InnoDB;
```

16.1 创建表

❖ 21.1.6 引擎类型

InnoDB

是一个可靠的事务引擎，它不支持全文本搜索

MEMORY

功能等同于**MyISAM**,但由于数据存储在内存中，速度很快

MyISAM

性能极高的引擎，它支持全文本搜索，但不支持事务处理（不指定是默认就是**MyISAM**）

16.2 更改表

- 1) 给表添加一个列

```
ALTER TABLE vendors
ADD vend_phone CHAR(20);
```

- 2） 删除一个列

```
ALTER TABLE vendors DROP COLUMN
vend_phone;
```

3) 增加外建

```
ALTER TABLE orderitems
ADD CONSTRAINT fk_orderitems_orders
FOREIGN KEY (order_num) REFERENCES
orders (order_num);
```

16.3 删除表

```
DROP TABLE customers2;
```

16.4重命名表

```
RENAME TABLE customer2 TO customers;
```